Measuring the Diversity of a Test Set With Distance Entropy

Qingkai Shi, Zhenyu Chen, *Member, IEEE*, Chunrong Fang, Yang Feng, and Baowen Xu, *Member, IEEE*

***Abstract*—Most existing metrics that we call *white-box* metrics, such as coverage metrics, require white-box information, like program structure information, and historical runtime information, to evaluate the fault detection capability of a test set. In practice, such white-box information is usually unavailable or difficult to obtain, which means they often cannot be used. In this paper, we propose a *black-box* metric, distance entropy, based on the diversification idea behind many published diversity-based techniques. Distance entropy provides a possible solution for test set evaluation when white-box information is not available. The empirical study illustrates that distance entropy can effectively evaluate test sets if the distance metric between tests is well defined. Meanwhile, distance entropy outperforms simple diversity metrics without increasing time complexity.**

***Index Terms*—Fault detection capability, diversity, metrics, minimum spanning tree.**

Acronyms and Abbreviations

MST	minimum spanning tree
SIR	software-artifact infrastructure repository
RQ	research question

Notations

T, T', T''	test set
t_i	test case
$\|T\|, n$	cardinal number of a set
V	vertex set of a graph
$(S, V-S)$	partition of set V
E, E'	edge set of a graph
e, e'	edge in a graph
$\mathcal{G}(T)$	test relationship graph of test set T
$\mathcal{G}'(T)$	sub-graph of $\mathcal{G}(T)$
$\mathcal{MST}(T), \mathcal{T}, \mathcal{T}'$	minimum spanning tree of $\mathcal{G}(T)$
$\mathcal{W}(\mathcal{G}(T))$	minimum weight set of $\mathcal{G}(T)$
w, w_i	weight of an edge in a graph
$H, H(T)$	distance entropy
W	sum of edge weights in $\mathcal{W}(\mathcal{G}(T))$
$\sum \mathcal{G}$	sum of all edge weights in graph $\mathcal{G}$
$\langle H_i, N_i \rangle, \langle H_i', N_i' \rangle$	pairs of distance entropy and the number of killed mutants
f	function mapping from a multi-set of edge weights to the value of diversity

Manuscript received August 29, 2014; revised December 11, 2014 and February 28, 2015; accepted May 17, 2015. Date of publication June 02, 2015; date of current version March 01, 2016. This work was supported in part by the National Basic Research Program of China (973 Program 2014CB340702) and in part by the National Natural Science Foundation of China (Grant No. 61170067, 61373013). Associate Editor: W. E. Wong. (*Corresponding authors: Baowen Xu and Zhenyu Chen.*)

The authors are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China (e-mail: sqk08@software.nju.edu.cn; zychen@software.nju.edu.cn; fy07@software.nju.edu.cn; fcr06@software.nju.edu.cn; bwxu@nju.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TR.2015.2434953

I. Introduction

It is important to evaluate the fault detection capability of a test set. Mutation analysis [1] is one of the most effective techniques to evaluate the fault detection capability of a test set. However, it needs to run tests on numerous mutants to obtain runtime information (i.e., the number of killed mutants), which is too expensive to be used in practice [2]. Like mutation analysis, most existing evaluation metrics [3], which we call *white-box* metrics, rely heavily on white-box information, such as program structure information, and historical runtime information, to measure the effectiveness of test sets; but such information is not always available.

For example, in some secret projects, we cannot analyze the source codes, or even binary codes. Therefore, a *black-box* metric that does not require white-box information is needed to evaluate the fault detection capability of a test set when white-box information is not available, and to guide black-box testing.

In recent years, many diversity-based techniques have gradually emerged from enormous research work, including [4], [5], and [6], which have shown that, in most cases, the effectiveness of a test set can benefit from its diversity. In software testing, high diversity requires that tests should be spread evenly in a specific space, such as the input space [5], or the output space [4]. A classic example using diversity is combination testing [7], one of whose theoretical bases is orthogonal experimental design [8] which requires the candidate tests to be evenly spread across the sampling space. Furthermore, one recent research paper showed that the effectiveness of a test set that contains evenly spreading tests is close to the theoretical upper bound of software testing effectiveness [9]. By theoretical analysis, A. Arcuri and L. Briand [10] also proved that, to maximize the fault detection possibility in one dimensional space, tests should be evenly spread across the entire space (see Fig. 1(a)). That is to

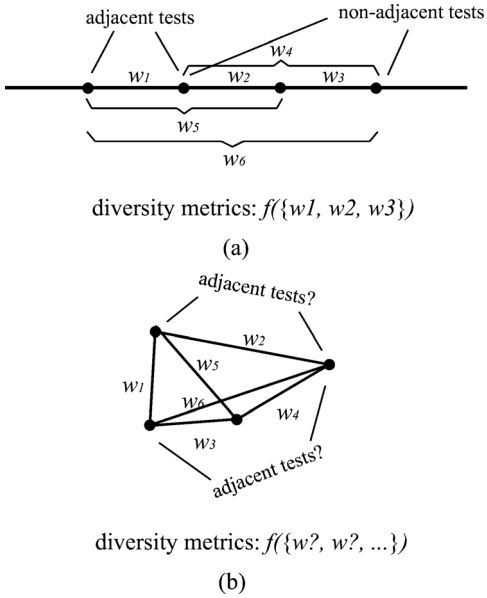

Fig. 1. (a) Example of an existing work in one dimensional space [10]. (b) Challenges in practice. w_i in the figure is the distances between tests.

say, to maximize the fault detection possibility in one dimensional space, tests should be spread as widely as possible, and the *distances between adjacent tests* should be the same. Following this work, there are two main challenges (see Fig. 1 (b)) to measure the diversity of a test set in practice, as follows.

Challenge 1: How do we define the adjacency relation between tests in practice?

Challenge 2: How do we make use of the distances between adjacent tests to measure the diversity of a test set?

For the first challenge, graph theory provides an important theoretical basis for us. It defines the adjacency relation as a symmetric binary relation between nodes that are linked with an edge. Therefore, we need to carefully select a suitable graph model to model a test set; at the same time, it also can describe the diversity property. In this paper, a minimum spanning tree is used because of its ability to measure the similarity and dissimilarity of the nodes in a graph [11].

For the second challenge, distance metrics like Euclidean distance have often been used [12]–[14] to measure the dissimilarity between two tests. However, these distance metrics cannot measure the dissimilarity of more than two tests. In contrast to distance, entropy has been commonly used to measure how evenly a set of entities are spread in a space, which inspires us to introduce entropy as a metric so that we can precisely measure the diversity of a test set, and provide insights into diversity in software testing.

In this paper, we first present the graph model of a test set. Based on the model, a lightweight black-box metric, namely distance entropy, is proposed to measure the diversity of a test set, thereby measuring its effectiveness.[1] Subsequently, an empirical study is conducted to illustrate the effectiveness and efficiency of distance entropy. The main contributions of this paper are as follows.

[1]Note that metrics for subjects other than test set effectiveness, e.g., Halstead metrics for program complexity evaluation, will not be discussed in the paper.

1) By modeling a test set as a minimum spanning tree, a metric, i.e., distance entropy, is proposed to measure the diversity, as well as the fault detection capability, of a test set, which provides a possible solution for test set evaluation without white-box information.
2) We demonstrate the effectiveness of distance entropy by comparing it with mutation analysis via an empirical study.
3) We also show that distance entropy outperforms metrics that naively use distance, which illustrates the value of the graph model for test sets.

The paper is organized as follows. Section II introduces the fundamental theories of distance entropy. Section III takes us from theoretical study to empirical study, in which we verify the efficacy of distance entropy. In Section IV, we discuss some practical issues. Section V describes the related work, and Section VI concludes this paper.

II. Distance Entropy

In this section, we first present the graph model for test sets. Based on the model, a novel black-box metric called distance entropy is proposed to measure the diversity of a test set. Distance entropy only depends on the weights of the edges (distances between tests, as described in Section II-D) in the graph. Note that distances between tests can be defined without any white-box information, thus being more flexible than those white-box metrics, especially when white-box information is not available. We also analyze the time complexity of distance entropy, which shows it is as efficient as some naïve diversity metrics.

A. Model

Given a test set T, to describe the relations between tests, we define the test relationship graph as a weighted undirected complete graph.

Definition 1 (Test Relationship Graph): A test relationship graph of a test set $T = \{t_1, t_2, \cdots, t_n\}$ $(n \geq 2)$ is a weighted undirected complete graph $\mathcal{G}(T) = \langle V, E \rangle$, in which $V = T$, and $E = \{(t_i, t_j) | t_i, t_j \in V, i \neq j\}$. The weight of each edge (t_i, t_j) is the distance between t_i and t_j.[2]

To precisely measure the diversity of a test set with acceptable cost, we should select a sub-set of representative edges from $\mathcal{G}(T)$. According to the intuition that similar tests, i.e., nodes that are connected by short edges in $\mathcal{G}(T)$, have negative effects on diversity, the short edges in $\mathcal{G}(T)$ should be evaluated preferentially. In other words, if all short edges are long enough, the diversity of the test set must be high. Following the intuition, for any partition of V, e.g., $(S, V - S)$, we only reserve the shortest edge between S and $V - S$ for evaluation. As a result, a sub-graph $\mathcal{G}'(T) = \langle V, E' \rangle$ is extracted from $\mathcal{G}(T)$ for diversity measurement. Exactly, the graph $\mathcal{G}'(T)$ is the minimum spanning tree (MST) of the test relationship graph $\mathcal{G}(T)$ [15]. For clarity, we denote $\mathcal{G}'(T)$ as $\mathcal{MST}(T)$, and define the weights of edges in $\mathcal{MST}(T)$ as a minimum weight set.

[2]We discuss how to define distance in Section II-D.

Definition 2 (Minimum Weight Set): The minimum weight set of a graph $\mathcal{G}(T)$, denoted as $\mathcal{W}(\mathcal{G}(T))$, is the multi-set[3] of the weights of edges in $\mathcal{MST}(T)$.

For example, an MST of a test relationship graph $\mathcal{G}(T)$ has four nodes, and three edges. The weights of the edges are 1, 2, 2, respectively. In this case, $\mathcal{W}(\mathcal{G}(T)) = \{1, 2, 2\}$.

According to the previous discussions, it is the weights in $\mathcal{W}(\mathcal{G}(T))$ that will be used to measure the diversity of a test set. Before defining the diversity metric, we highlight the three important properties of MST as below that make it suitable to measure diversity.

- An MST is a connected graph so that all tests in a test set can be taken into consideration.
- Any edge e in an MST is the shortest edge between two partitions of the test relation graph.
- Given a test set, and its relationship graph, the minimum weight set is unique, which ensures that the diversity metric of a test set is a unique value.

The properties are straightforward, except for the third one. Gallager *et al.* [16] proved that, if each edge in a graph has a distinct weight, then there will be only one unique minimum spanning tree. However, there may exist edges with the same weights in practice (i.e., there are some pairs of tests that have the same distance), which will result in a graph having multiple MSTs. We prove the uniqueness of the minimum weight set as below.

Theorem 1: Given a test relationship graph $\mathcal{G}(T)$, the minimum weight set $\mathcal{W}(\mathcal{G}(T))$ is unique.[4]

B. Definition

Given a test set T, and its minimum weight set $\mathcal{W}(\mathcal{G}(T))$, in this section, we define $H(T) = f(\mathcal{W}(\mathcal{G}(T)))$ as the diversity metric. To measure diversity precisely, $H(T)$ should have the maximum iff both the *even spreading condition* and the *wide spreading condition* are satisfied.

- *Even spreading condition*: $\forall w_i, w_j \in \mathcal{W}(\mathcal{G}(T)), w_i = w_j$. That is to say, tests should spread evenly in a specific space.
- *Wide spreading condition*: $\forall w \in \mathcal{W}(\mathcal{G}(T)), w$ should be as large as possible. That is to say, tests should spread widely in a specific space.

Following the definition of Shannon entropy [17], and based on the above two conditions, we define the black-box metric, distance entropy, as below.

Definition 3 (Distance Entropy): The distance entropy H of a test set T is defined based on the minimum weight set, i.e., $\mathcal{W}(\mathcal{G}(T))$:

$$H(T) = W \times (-\sum_{w \in \mathcal{W}(\mathcal{G}(T))} \frac{w}{W} \ln \frac{w}{W}), \tag{1}$$

in which

$$W = \sum_{w \in \mathcal{W}(\mathcal{G}(T))} w.$$

[3]In mathematics, a multi-set is a generalization of set. Elements in a multi-set are allowed to appear more than once.

[4]Please find all proofs in the Appendix.

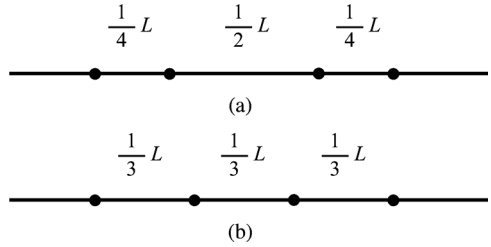

Fig. 2. An example to illustrate that at least when $W = L$, which is fixed, distance entropy is much better. The black dots stand for tests.

TABLE I
RESULTS OF COMPARISON

Metrics	Results of (a)	Results of (b)	Conclusions
$\sum \mathcal{G}$	$3.5L$	$3.33L$	(a) is better
W	$1L$	$1L$	(a) = (b)
H	$1.04L$	$1.10L$	(b) is better

$\sum \mathcal{G}$: sum of all distances between different tests.
W: sum of all weights in the minimum weight set.
H: distance entropy.

Distance entropy (1) consists of two parts. The first one is W, which ensures that the *wide spreading condition* is satisfied. That is, even though tests are evenly spread in a space, to make the diversity high, tests must be spread widely. The other part of distance entropy is consistent with Shannon entropy, which guarantees that the *even spreading condition* is satisfied when W is fixed, which has been proved in [17].

Distance entropy is logically dependent on the definition of distance. So one question is whether distance entropy outperforms some simple metrics (e.g., sum of distances). It is difficult to verify the question theoretically in general. Hence, this question will be studied empirically in Section III. The following example illustrates that, at least when the value of W is fixed, distance entropy is much better in the one-dimensional space. The metrics used to compare with distance entropy are two simple ones that use distance to measure diversity:

- $\sum \mathcal{G}$ is the sum of all weights in the test relationship graph $\mathcal{G}(T)$, and
- W is the sum of all weights in the minimum weight set (see Definition 3).

In Fig. 2, based on Euclidean distance, part (b) is obviously better than part (a) according to the theorems in [10], because tests in (b) are spread more evenly than those in (a). We calculate the value of three metrics to measure the diversity. The results are shown in Table I. Obviously, only when using distance entropy can we make a correct conclusion (i.e., (b) is better than (a)). To verify the effectiveness of distance entropy in practice, we will empirically compare H, W, and $\sum \mathcal{G}$ in Section III.

C. Complexity

We prove the following theorem to show that the time complexity of distance entropy is asymptotically equal to W, and $\sum \mathcal{G}$, two naive metrics. The whole process of using distance entropy to evaluate the diversity of a test set is shown in Fig. 3.

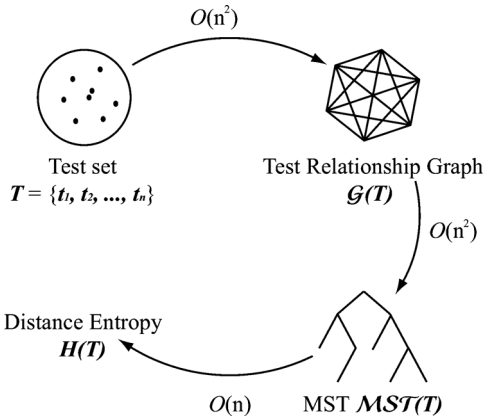

Fig. 3. The process of using distance entropy to measure the diversity of a test set.

Theorem 2: Given a test set $T(|T| = n)$, the time complexity of using distance entropy to measure its diversity is $O(n^2)$, which is asymptotically equal to the time complexity of using W and $\sum \mathcal{G}$.

D. Distance Calculation

In this subsection, we discuss how to define an appropriate distance metric, which will drive the distance entropy metric more effectively. Generally, distance metrics can be defined according to data types. For example, Euclidean distance can be used for numeric data, edit distance can be used for character strings, and so forth. However, these typical relations between data types and distance metrics are not always useful. For example, Euclidean distance is not suitable when dimensions are incomparable, and on different scales.

To define the distance between tests, a more practical, intelligent way is to learn a distance metric from existing data. Recently, there has been appreciable research on distance metric learning [18], which can be divided into two categories: supervised distance metric learning, and unsupervised distance metric learning. Previous work [18] has illustrated that a learned distance metric can significantly benefit the accuracy of similarity calculation compared to the standard Euclidean distance.

Therefore, we can choose different methods to define distance metrics to drive distance entropy according to actual situations.

III. Empirical Study

In the previous sections, we investigated some properties of distance entropy in theory. Here, the following research questions (RQ) are studied empirically to show the efficacy of distance entropy.

RQ1 Does a high distance entropy imply a high bug detection rate, when white-box information is unavailable?

RQ2 Does distance entropy outperform those simple diversity metrics (i.e., W, and $\sum \mathcal{G}$)?

A. Subject Programs

We use five real, frequently-used programs as the subject programs of our empirical study. The five programs come from the Software-artifact Infrastructure Repository (SIR) [19], which has been widely used in the research of testing techniques. The information of the five subject programs is shown in Table II.

TABLE II
Subject Programs

Program	Version	Lines of Code	Size of Test Pool
gzip	1.1.2	5680	214
grep	2.4	10068	809
flex	2.5.1	10459	567
sed	4.0.7	14427	363
make	3.76.1	35545	793

The general criteria that we use to select programs were that, first, they must be frequently-used programs. The programs listed in Table II are used by countless Linux programmers every day. Second, the scale of programs must be suitable. By considering both the lines of programs and the sizes of test pools, we select the largest programs possible from the SIR.

All these test pools were constructed following different algorithms or criteria, each of which may represent a test suite constructed by a software engineer [19].

B. Methodology

Although these benchmark programs contain some seeded or real bugs in the SIR, the number of these bugs is too small to obtain statistical significance in the experiments. Instead, mutation analysis is used.

1) Mutation Analysis: As shown before, mutation analysis is a widely used technique to evaluate the effectiveness of test sets [20]. Lots of research work supports that mutation analysis can provide a good indication of the fault detection capability of a test set [21]. In our empirical study, mutation analysis is used to evaluate the efficacy of distance entropy by evaluating the fault detection capability of test sets. Here, the number of killed mutants during mutation analysis is used as a metric of fault detection capability.

In our experiment, `mutate.py` [43] is used to mutate benchmark programs. `mutate.py` is a python script which can be used to mutate programs directly. It is easy-to-use, and contains all kinds of sufficient mutation operators [22]. These mutation operators include the absolute value insertion, which forces each arithmetic expression to take on a zero value, a positive value, and a negative value; the arithmetic operator replacement, which replaces each arithmetic operator with every syntactically legal operator; the logical connector replacement, which replaces each logical connector with several kinds of logical connectors; the relational operator replacement, which replaces relational operators with other relational operators; and the unary operator insertion, which inserts unary operators in front of expressions. These five mutation operators suffice to effectively implement mutation testing [22]. Even so, `mutate.py` also contains some other mutation operators, like the source constant replacement, which replaces constants in the source codes with other constants.

After mutation, and manually removing equivalent mutants, we randomly obtained five hundred executable mutants for each program. Note that random mutant selection is not inferior to operator-based mutant selection [23]. By running all tests of

TABLE III
RESULTS FOR RQ1 AND RQ2

Size of Test Sets	flex			grep			gzip			make			sed		
	$\boldsymbol{H}$	$\sum \mathcal{G}$	W	$\boldsymbol{H}$	$\sum \mathcal{G}$	W	$\boldsymbol{H}$	$\sum \mathcal{G}$	W	$\boldsymbol{H}$	$\sum \mathcal{G}$	W	$\boldsymbol{H}$	$\sum \mathcal{G}$	W
5	0.86	0.72	0.75	0.85	0.80	0.80	0.79	0.78	0.80	0.88	0.80	0.76	0.62	0.55	0.40
10	0.80	0.73	0.77	0.96	0.77	0.78	0.77	0.71	0.77	0.90	0.77	0.80	0.40	0.50	0.43
15	0.95	0.63	0.80	0.91	0.79	0.81	0.74	0.66	0.68	0.90	0.81	0.85	0.79	0.42	0.44
20	0.91	0.68	0.83	0.96	0.82	0.88	0.68	0.68	0.65	0.92	0.76	0.73	0.88	0.62	0.66
25	0.89	0.54	0.88	0.89	0.82	0.88	0.70	0.50	0.50	0.91	0.77	0.77	0.91	0.63	0.61
30	0.92	0.70	0.78	0.94	0.87	0.90	0.56	0.58	0.60	0.95	0.69	0.77	0.90	0.59	0.77
40	0.98	0.78	0.86	0.94	0.76	0.86	0.56	0.72	0.55	0.96	0.71	0.80	0.86	0.59	0.69
50	0.96	0.80	0.88	0.93	0.73	0.85	0.55	0.55	0.52	0.95	0.78	0.82	0.92	0.70	0.71

each program on all its mutants, and comparing the outputs with those obtained from the original version, a mutation matrix is established. It contains information about whether a test can kill a mutant or not. To evaluate the effectiveness of a test set, we only need to count the number of killed mutants by referring to the mutation matrix.

2) Test Set Construction: For each program, we randomly select tests from its test pool. We repeat the process by selecting different sizes (5, 10, 15, 20, 25, 30, 40, 50) of test sets. For each size, we randomly select 2500 test sets. We chose 50 as the upper bound for the experiment because we observed that a test set of that size usually can achieve the upper bound of fault detection capability of the test pool.

3) Distance Definition: To calculate distance entropy, as well as the simple metrics W and $\sum \mathcal{G}$, we should first calculate the distances between tests. Therefore, a proper distance metric is important, otherwise we may get an illusion that distance entropy does not work.

We have discussed a variety of distance metrics in Section II-D. We also showed that we can learn a distance metric by data mining techniques. Considering that the tests for all five benchmark programs are character strings, we simply choose the edit distance for the tests of all these five programs to show the general effectiveness of distance entropy.

C. Addressing RQ1

With the mutation matrix, types of test sets, and a distance definition in hand, we conduct the **RQ1** as below.

1) For each test set i with a specific size, its distance entropy H_i, and the number of killed mutants N_i are calculated.
2) Sort the 2500 test sets by distance entropy, and we get a list of $\langle H_i, N_i \rangle$ in ascending order of distance entropy.
3) Compute the mean values of distance entropy, and the number of killed mutants for every 100 neighboring items. A list of $\langle H'_j, N'_j \rangle$ is obtained.
4) Draw each point $\langle H'_j, N'_j \rangle$ in a rectangular coordinate system, and fit a straight line for these scatter dots using the least squares method. The Pearson correlation coefficient [24] is calculated to evaluate the goodness of fit.

The results are shown in Table III, which shows that the correlation between distance entropy and the number of killed mutants is very strong, except for the program `gzip`, which is a data compression program. The observation shows that distance entropy can be used to evaluate the effectiveness of test sets, as well as mutation analysis.

Why does distance entropy not work well for `gzip`? To find the reason, the tests and the user manual of `gzip` are investigated. We find that the outputs of `gzip` are heavily determined by the files which `gzip` works on. And when we use the edit distance to calculate the dissimilarity between two tests of `gzip`, the contents of the files are ignored. In contrast to `gzip`, the other programs depend more on the options and arguments in the command line strings, which are more sensitive to the edit distance.

D. Addressing RQ2

In Section II, an example is used to illustrate that distance entropy H is better than $\sum \mathcal{G}$ and W in a one-dimensional space. In this subsection, we will verify the conclusion in practice by empirical study. We repeat the process of the experiment for the **RQ1** by using $\sum \mathcal{G}$ and W to measure the diversity instead of the distance entropy.

The Pearson correlation coefficients are also shown in Table III, which shows that the coefficients of $\sum \mathcal{G}$ and W are smaller than distance entropy in almost all cases. To make the conclusion more visible, we give the plots of $\sum \mathcal{G}$ and W for each program (size $= 10$) in Fig. 4. As shown in Fig. 4, $\sum \mathcal{G}$ and W are not strongly related to the number of killed mutants; and for the program `sed`, the points are far from a monotone curve.

For metrics $\sum \mathcal{G}$ and W, they both simply use distance to measure diversity. Because distance can only be used to measure the difference between two tests, but not the distribution of all tests, simply using distance to measure diversity is not suitable. For example, W only takes the sum of the weights of an MST into consideration, but does not consider the distribution of all tests. Similarly, $\sum \mathcal{G}$ is also not a good metric.

In summary, distance entropy not only outperforms $\sum \mathcal{G}$ and W, but also its time complexity is asymptotically equal to them (Theorem 2).

IV. DISCUSSION

A. Threats to Validity

Our empirical study of five real-world, frequently-used programs allow us to make some important, interesting conclusions. Like any other empirical studies, some threats to the validity still exist. This section is concluded by considering the internal, external, and construct threats to the validity of the empirical study.

1) Internal Threats: Threats to internal validity may come from errors in our implementation of tools used to collect data and do statistics. To minimize this kind of threat, those tools are

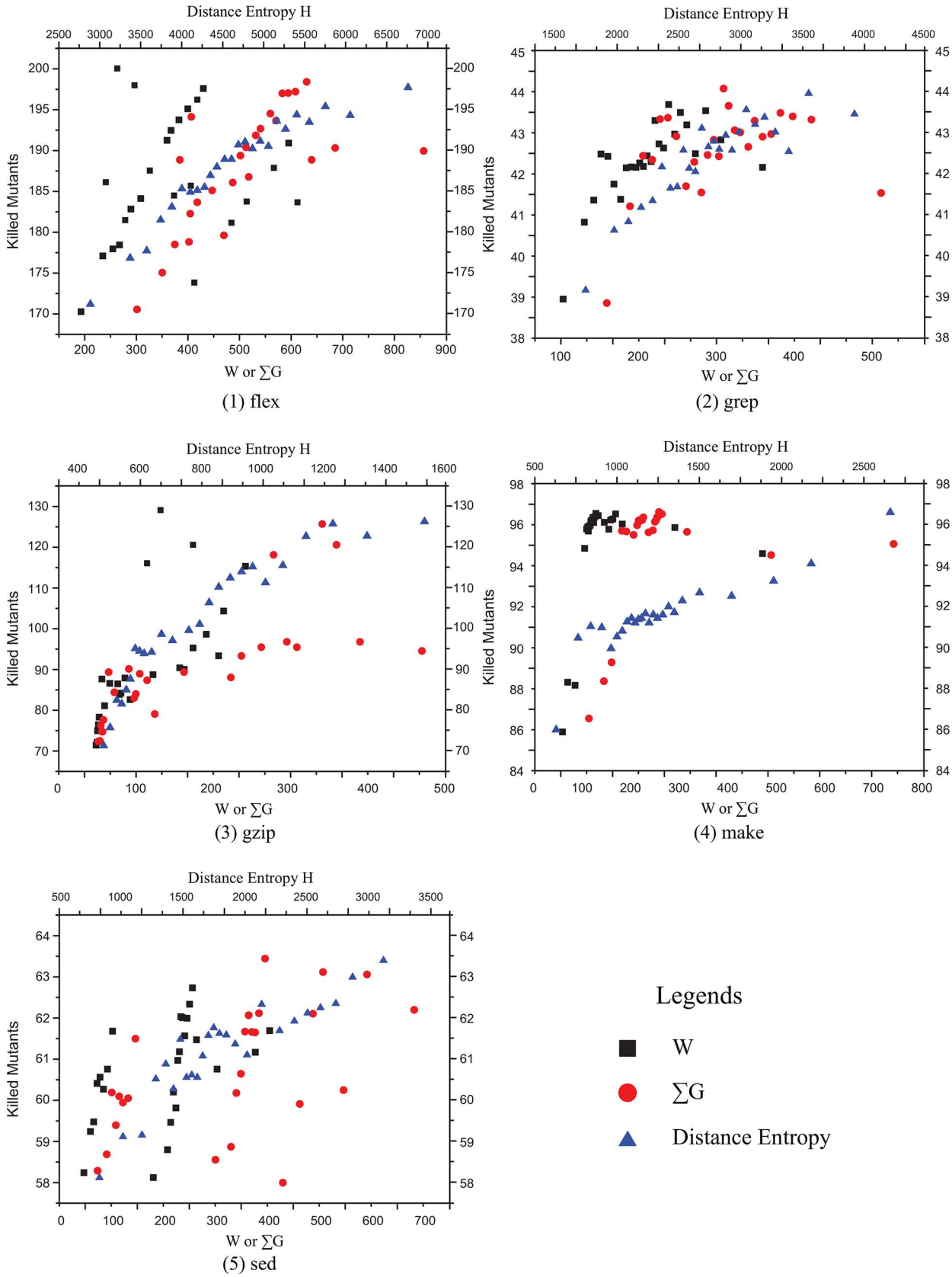

Fig. 4. Results of RQ2.

thoroughly tested with a great deal of small examples. In addition, we try our best to review the codes of those tools manually. Besides, we try our best to use open source tools, and refer to their documents. For example, the mutation tool used in our

experiment requires us to normalize the source codes of the subject programs before using it to produce mutants.

Last but not least, we also ensure the correctness of the reported results by repeating our experiment several times for each program. The consistent results of these repetitions make us more confident of our conclusions.

2) External Threats: There are also threats to the external validity of our empirical study. The main external threat is that the five programs cannot stand for a wide range of all possible programs. To mitigate this threat, the programs we selected are all real-world, frequently-used programs. And we downloaded these programs from SIR, which is a widely used benchmark source for software testing.

One of the other external threats may be the tool we used to produce mutants. There are many mutation tools [1] because mutation analysis has been widely studied. To minimize the external threat, by manual review, we ensure that all kinds of sufficient mutant operators [22] are used in our experiment. It is a time-consuming effort, but it increases the reliability of our conclusions.

3) Construct Threats: There are many alternative definitions of distance. In empirical studies, edit distance is used to drive distance entropy, because the tests in our benchmarks are character strings, and edit distance is the most widely-used metric for the comparison of two character strings.

B. Distance Entropy in Risk-Driven Projects

We usually can expect that about 80% of the bugs come from 20% of the modules in a program. Therefore, some readers may be concerned that distance entropy, which requires tests evenly spread, may not work on such programs.

On the one hand, in most situations, we cannot know the distribution of the software bugs; hence, we have to suppose that the bugs are evenly spread, because every part of the codes may have bugs. On the other hand, even though we know the exact part that contains most of the bugs in a program, we still can use distance entropy for that part of the program, because we still do not know how the bugs are distributed in that part.

C. Test Sets With Different Sizes

In the current version, distance entropy can only be used to compare test sets with the same size. Here, we provide a solution for test sets with different sizes.

Given a set T, $|T| = n$, and a set T', $|T'| = n + k(k > 0)$, suppose $T'' \subseteq T'$, and $|T''| = |T| = n$. We can first compare T'' and T using distance entropy, and then add tests in $T' - T''$ into the winner (i.e., T'' if $H(T'') > H(T)$, T otherwise), thereby obtaining a better test set (i.e., T' if $H(T'') > H(T)$, $T \cup (T' - T'')$ otherwise).

V. Related Work

In this section, we summarize some related work by considering white-box metrics, diversity-based black-box metrics & techniques, and the entropy used in software testing. We also discuss the difference between the existing research and ours.

A. White-Box Metrics

Coverage based metrics are the most commonly used white-box metrics, which requires white-box information, e.g., the historical runtime information of tests and program structure information, to evaluate test sets. Structure coverage, including branch coverage, condition coverage, etc., is one of the most commonly used criteria to guide test generation and selection [3]. In some cases, it also can be used to evaluate the effectiveness of test sets. However, there are many disputes about coverage criteria [25], [26]. The most recent report [27] showed that coverage is not strongly correlated with test suite effectiveness.

Mutation adequacy score, which is produced by mutation analysis [1], is another kind of coverage-based adequacy criterion. However, although many techniques have been proposed to reduce the costs of mutation analysis (e.g., [28]) and detect equivalent mutants (e.g., [29]), the two issues are still the main bottlenecks.

As white-box metrics take advantage of white-box information, it must be more effective than distance entropy, which does not need any white-box information. However, we do not think such a fact degrades the contribution, because white-box metrics will cease to be effective when white-box information is unavailable. In this paper, we provide a possible solution for the scenario where no white-box information can be used, and we have shown that the black-box metric, distance entropy, could work well without such white-box information as long as we define the distance suitably.

B. Diversity-Based Black-Box Metrics & Techniques

There have been some pieces of work studying how to measure the diversity of tests. Some of them [13], [30] in fact utilize distance metrics or similarity metrics instead of a real diversity metric. [31] presented simple distribution metrics, e.g., dispersion and discrepancy, to measure the diversity of a test set. However, they do not have enough ability to distinguish between different test sets. For instance, different test sets which can be compared by distance entropy may have the same dispersion value, because dispersion only takes the maximin distance [32] into consideration.

On diversity-based techniques, adaptive random testing [12] is an extended version of random testing whose effectiveness is close to the upper bound of software testing effectiveness [9]. Ledru *et al.* [33], [34] used string distance instead of Euclidean distance to compare test data. Recently, diversity in the output space also has been utilized in web applications [4]. Besides, many diversity-based techniques have been proposed for model-based testing techniques [6], [35]–[37], and event sequence-based testing techniques [38].

C. Entropy

Information theory was famously founded by Shannon [17], and has been widely used in software testing since Agrawal [39] used information theory to test digital circuits in 1981. Even so, we were the first to use entropy to measure the diversity of a test set.

In recent years, entropy has been widely used in fault localization. H. Cheng *et al.* [40] used entropy and information gain

to mine discriminative graphs to identify bug signatures. Roychowdhury *et al.* [41] studied a family of generalized entropy models, and presented a fault localization technique based on generalized mutual information. Shannon entropy is used by Yoo *et al.* [42] to prioritize tests based on their ability to reduce fault localization entropy for fault localization.

VI. Conclusion

In this paper, we proposed a novel black-box metric, distance entropy, to measure the diversity and effectiveness of test sets without white-box information. By modeling a test set as an MST, we study some of its important properties theoretically. We show that, although distance entropy is a function of distance, it not only outperforms some simple metrics, but also has an asymptotically equivalent time complexity with those simple metrics. The empirical study showed that distance entropy, which does not use any white-box metric, has consistent results with mutation analysis when used to measure the effectiveness of different test sets with the same size. We believe that distance entropy is a good potential metric to measure the diversity and effectiveness of test sets where white-box information is usually unavailable.

Appendix

The Proof for Theorem 1.

Proof:

1. If all weights in E are unique, then the MST of $\mathcal{G}$ will be unique. Therefore, the minimum weight set is unique.
2. If the weights in E are not unique, $\mathcal{G}$ will contain more than one MST. Suppose $\mathcal{T}$, and $\mathcal{T}'$ are two different MSTs of $\mathcal{G}$. We only need to prove that, if an edge e in $\mathcal{T}$ is not in $\mathcal{T}'$, there must exist an edge e' in $\mathcal{T}'$ but not in $\mathcal{T}$, and their weights are the same.

Let e be an edge that is in $\mathcal{T}$ but not in $\mathcal{T}'$. As $\mathcal{T}'$ is an MST, $\{e\} \cup \mathcal{T}'$ must contain a cycle. Then $\mathcal{T}'$ must contain at least one edge e' that is not in $\mathcal{T}$ and lies on the cycle. Assume the weight of e is not equal to e'; without loss of generality, suppose the weight of e is less than that of e'. Replacing e' with e in $\mathcal{T}'$ leads to the result that the spanning tree $\{e\} \cup \mathcal{T}' - \{e'\}$ has a smaller weight compared to $\mathcal{T}'$, which contradicts our premise.

Therefore, the weight of e must be equal to e', and the minimum weight set is unique. □

The Proof for Theorem 2.

Proof: Suppose the test relationship graph of T is $\mathcal{G}(T) = \langle V, E\rangle$, and its MST is $\mathcal{MST}(T) = \langle V, E'\rangle$. As $|T| = n$, then $|V| = n$, $|E| = n(n-1)/2 = O(n^2)$, and $|E'| = n - 1 = O(n)$.

1. To calculate the distance entry, first, we need to calculate all distances between tests to establish the test relationship graph $\mathcal{G}(T)$. The time complexity is

$$O(|E|) = O(n^2).$$

Second, an MST is generated from $\mathcal{G}(T)$. According to [15], using a Fibonacci heap, the time complexity of the Prim algorithm is

$$O(|E| + |V| \lg |V|) = O(n(n-1)/2 + n \lg n) = O(n^2).$$

Third, according to the definition, the time complexity to calculate distance entropy is

$$O(|E'|) = O(n).$$

Therefore, the overall time complexity is

$$O(n^2) + O(n^2) + O(n) = O(n^2).$$

2. For the metric W, like distance entropy, the time complexity to produce an MST is $O(n^2)$. With the MST, the time complexity to calculate W is $O(|E'|) = O(n)$. Therefore, the overall time complexity of W is also $O(n^2)$.
3. For the metric $\sum \mathcal{G}$, i.e., the sum of all distances, because $|E| = n(n-1)/2 = O(n^2)$, the time complexity of $\sum \mathcal{G}$ is $O(n^2)$.

   In conclusion, the time complexity to calculate distance entropy, W, and $\sum \mathcal{G}$ is the same.

□

Acknowledgment

The authors wish to express deep appreciation to the anonymous reviewers for their insightful and constructive comments on an early draft of this paper.

References

[1] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Trans. Softw. Eng.*, vol. 37, no. 5, pp. 649–678, 2011.
[2] B. Grun, D. Schuler, and A. Zeller, "The impact of equivalent mutants," in *Proc. IEEE Int. Conf. Software Testing, Verification, and Validation Workshops (ICSTW'09)*, 2009, pp. 192–199.
[3] P. Ammann and J. Offutt, *Introduction to Software Testing*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
[4] N. Alshahwan and M. Harman, "Augmenting test suites effectiveness by increasing output diversity," in *Proc. 34th Int. Conf. Software Engineering (ICSE'12)*, 2012, pp. 1345–1348.
[5] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and T. Tse, "Adaptive random testing: The art of test case diversity," *J. Syst. Softw.*, vol. 83, no. 1, pp. 60–66, 2010.
[6] H. Hemmati, A. Arcuri, and L. Briand, "Achieving scalable model-based testing through test case diversity," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 1, pp. 6:1–6:42, 2012.
[7] M. Grindal, J. Offutt, and S. Andler, "Combination testing strategies: A survey," *Softw. Test., Verif., Rel.*, vol. 15, no. 3, pp. 167–199, 2005.
[8] W. Cochran and G. Cox, *Experimental Designs*. New York, NY, USA: Wiley, 1957.
[9] T. Y. Chen and R. Merkel, "An upper bound on software testing effectiveness," *ACM Trans. Softw. Eng. Methodol.*, vol. 17, no. 3, pp. 1–27, 2008.
[10] A. Arcuri and L. Briand, "Adaptive random testing: An illusion of effectiveness?," in *Proc. 20th Int. Symp. Software Testing and Analysis (ISSTA'11)*, 2011, pp. 265–275.
[11] Y. Zhou, O. Grygorash, and F. Thomas, "Clustering with minimum spanning trees," *Int. J. Artif. Intell. Tools*, vol. 20, no. 01, pp. 139–177, 2011.
[12] T. Y. Chen, H. Leung, and I. Mak, "Adaptive random testing," *Adv. Comput. Sci.*, vol. 3321, no. 1, pp. 320–329, 2004.
[13] R. Feldt, R. Torkar, T. Gorschek, and W. Afzal, "Searching for cognitively diverse tests: Towards universal test diversity metrics," in *Proc. 2008 IEEE Int. Conf. Software Testing Verification and Validation Workshop (ICSTW'08)*, 2008, pp. 178–186.
[14] S. Yoo, M. Harman, P. Tonella, and A. Susi, "Clustering test cases to achieve effective & scalable prioritisation incorporating expert knowledge," in *Proc. 18th Int. Symp. Software Testing and Analysis (ISSTA'09)*, 2009, pp. 201–211.
[15] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2001.

[16] R. Gallager, P. Humblet, and P. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Trans. Program. Lang. Syst.*, vol. 5, no. 1, pp. 66–77, 1983.
[17] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 1, pp. 379–423, 623–656, 1948.
[18] L. Yang and R. Jin, Distance Metric Learning: A Comprehensive Survey Michigan State Univ., East Lansing, MI, USA, Tech. Rep., 2006.
[19] H. Do, S. G. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empir. Softw. Eng.: Int. J.*, vol. 10, no. 4, pp. 405–435, 2005.
[20] R. DeMillo, R. Lipton, and F. Sayward, "Hints on test data selection: Help for the practicing programmer," *IEEE Comput.*, vol. 11, no. 4, pp. 34–41, 1978.
[21] J. Andrews, L. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?," in *Proc. 27th Int. Conf. Software Engineering (ICSE'05)*, 2005, pp. 402–411.
[22] A. Offutt, A. Lee, G. Rothermel, R. Untch, and C. Zapf, "An experimental determination of sufficient mutant operators," *ACM Trans. Softw. Eng. Methodol.*, vol. 5, no. 2, pp. 99–118, 1996.
[23] L. Zhang, S. Hou, J. Hu, T. Xie, and H. Mei, "Is operator-based mutant selection superior to random mutant selection?," in *Proc. 32nd ACM/IEEE Int. Conf. Software Engineering (ICSE'10)*, 2010, pp. 435–444.
[24] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Mahwah, NJ, USA: Lawrence Erlbaum, 1988.
[25] A. Namin and J. Andrews, "The influence of size and coverage on test suite effectiveness," in *Proc. 18th Int. Symp. Software Testing and Analysis (ISSTA'09)*, 2009, pp. 57–68.
[26] K. Aaltonen, P. Ihantola, and O. Seppälä, "Mutation analysis vs. code coverage in automated assessment of students' testing skills," in *Proc. ACM Int. Conf. Companion Object Oriented Programming Systems Languages and Applications Companion (OOPSLA'10)*, 2010, pp. 153–160.
[27] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," in *Proc. 36th Int. Conf. Software Engineering (ICSE'14)*, 2014, pp. 435–445.
[28] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid, "Regression mutation testing," in *Proc. 2012 Int. Symp. Software Testing and Analysis (ISSTA'12)*, 2012, pp. 331–341.
[29] D. Schuler and A. Zeller, "Covering and uncovering equivalent mutants," *Softw. Test., Verif., Rel.*, vol. 23, no. 5, pp. 353–374, 2013.
[30] B. Nikolik, "Test diversity," *Inf. Softw. Technol.*, vol. 48, no. 11, pp. 1083–1094, 2006.
[31] O. T. C. D. of Adaptive Random Testing, "On test case distributions of adaptive random testing," in *Proc. 19th Int. Conf. Software Engineering and Knowledge Engineering (SEKE'07)*, 2007, pp. 141–144.
[32] M. Johnson, L. Moore, and D. Ylvisaker, "Minimax and maximin distance designs," *J. Statist. Plan. Inference*, vol. 26, no. 2, pp. 131–148, 1990.
[33] Y. Ledru, A. Petrenko, and S. Boroday, "Using string distances for test case prioritisation," in *Proc. 2009 IEEE/ACM Int. Conf. Automated Software Engineering (ASE'09)*, 2009, pp. 510–514.
[34] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," *Autom. Softw. Eng.*, vol. 19, no. 1, pp. 65–95, 2012.
[35] H. Hemmati, A. Arcuri, and L. Briand, "Reducing the cost of model-based testing through test case diversity," in *Proc. 22nd IFIP WG 6.1 Int. Conf. Testing Software and Systems (ICTSS'10)*, 2010, pp. 63–78.
[36] H. Hemmati, A. Arcuri, and L. Briand, "Empirical investigation of the effects of test suite properties on similarity-based test case selection," in *Proc. 2011 4th IEEE Int. Conf. Software Testing, Verification and Validation (ICST'11)*, 2011, pp. 327–336.
[37] H. Hemmati and L. Briand, "An industrial investigation of similarity measures for model-based test case selection," in *Proc. 2010 IEEE 21st Int. Symp. Software Reliability Engineering (ISSRE'10)*, 2010, pp. 141–150.
[38] P. Brooks and A. Memon, "Introducing a test suite similarity metric for event sequence-based test cases," in *Proc. 2009 IEEE Int. Conf. Software Maintenance (ICSM'09)*, 2009, pp. 243–252.
[39] V. Agrawal, "An information theoretic approach to digital fault testing," *IEEE Trans. Comput.*, vol. C-30, no. 8, pp. 582–587, 1981.
[40] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan, "Identifying bug signatures using discriminative graph mining," in *Proc. 18th Int. Symp. Software Testing and Analysis (ISSTA'09)*, 2009, pp. 141–152.
[41] S. Roychowdhury and S. Khurshid, "A family of generalized entropies and its application to software fault localization," in *Proc. 6th IEEE Int. Conf. Intelligent Systems (IS'12)*, 2012, pp. 368–373.
[42] S. Yoo, M. Harman, and D. Clark, "Fault localization prioritization: Comparing information theoretic and coverage based approaches," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 3, pp. 19:1–19:29, 2013.
[43] Tools for mutation analysis [Online]. Available: https://github.com/qingkaishi/mutation.git

Qingkai Shi got a B.Eng. from Nanjing University in June 2012. He is a postgraduate student in the Software Institute, Nanjing University, under the supervision of Prof. Zhenyu Chen and Prof. Baowen Xu.

His research interest is in program analysis and testing. He visited the Hong Kong University of Science and Technology as a visiting postgraduate student under the supervision of Prof. Charles Zhang from 2013 to 2014.

Zhenyu Chen (M'09) received his bachelor and Ph.D. in mathematics from Nanjing University in 2001 and 2006, respectively.

He is currently an Associate Professor at the Software Institute, Nanjing University. He worked as a Postdoctoral Researcher at the School of Computer Science and Engineering, Southeast University, China. His research interests focus on software analysis and testing. He has about 70 publications in journals and proceedings including TOSEM, TSE, JSS, SQJ, IJSEKE, ISSTA, ICST, QSIC, etc. He has served as PC co-chair of QSIC 2013, AST2013, IWPD2012, and the program committee member of many international conferences. He has won research funding from several competitive sources such as NSFC.

Chunrong Fang is a Ph.D. student at the Software Institute, Nanjing University and a visiting Ph.D. student at the University of Maryland, College Park under the supervision of Prof. Atif Memon. He obtained his B.Eng. with honours from Nanjing University.

He has published extensively at premium conferences and in journals such as JSS and SEKE. His current research interest focuses on quality assurance for mobile programs.

Yang Feng received his B.Eng. and M.Eng. from Nanjing University. He is a Ph.D. student at the Software Institute at Nanjing University.

With the guidance of Profs. Baowen Xu and Zhenyu Chen, he focuses on building more intelligent tools to assist software engineering tasks. His current research interests include software testing, debugging, program analysis, and program comprehension. He also has a strong interest in crowdsourcing techniques, especially crowdsourcing testing, debugging, and task assignment strategies.

Baowen Xu (M'98) received the B.S., M.S., and Ph.D. degrees in computer science from Wuhan University, Huazhong University of Science and Technology, and Beihang University in 1982, 1985 and 2002, respectively.

He is a Professor in the Department of Computer Science and Technology at Nanjing University. His main research interests are programming languages, software testing, software maintenance, and software metrics. He has published extensively in premiere software engineering conferences and journals such as TOSEM, TSE, JSS, ICST, QSIC, COMPSAC, etc.